TP7 : introduction au projet Sudoku

Le projet consiste à développer des outils logiciels autour du jeu de Sudoku. L'article que lui consacre Wikipedia (`http://fr.wikipedia.org/wiki/Sudoku`) contient de nombreuses informations intéressantes.

Les fonctionnalités qui devront être programmées comprennent au minimum :

- résoudre une grille,
- implanter un ensemble de règles le plus complet possible de manière à ce que la résolution d'une grille s'accompagne d'une description et d'une explication de chaque étape.

Cette liste pourra être complétée par la suite, par exemple : production de grilles, décider si 2 grilles sont équivalentes aux permutations de chiffres, lignes et colonnes près, proposer et implanter une classification des grilles selon la difficulté de leur résolution. Je ne demanderai pas de fonctionnalités concernant l'interface utilisateur (vous pouvez en développer une si cela vous intéresse - après tout le reste). L'objectif essentiel de ce module est de renforcer vos connaissances en matière d'algorithmique, de programmation en C, et de vous donner de bonnes habitudes de développement, parmi lesquelles :

- la *programmation modulaire* : le programme devra comprendre plusieurs modules logiquement constitués et un fichier Makefile devra être fourni afin de construire l'exécutable du projet ;
- la production d'un *code source clair et commenté* ;
- la production d'un *rapport* accompagnant votre programme : il devra contenir une *introduction* au jeu et aux logiciels qui lui sont dédiés, un *mode d'emploi* de votre programme destiné à l'utilisateur décrivant ses fonctionnalités et la manière de l'utiliser, une *section technique* décrivant les structures de données et les solutions techniques que vous avez retenues ainsi qu'une description des algorithmes principaux de votre logiciel.
- la capacité à travailler en équipe sur un même projet.

Je propose les structures suivantes :

```
/* Grille 9x9 décomposée en 9 régions de taille 3x3 */
#define DIM       9
#define DIM_Region 3

/* une case comprend 5 champs :
- un tableau associant la valeur 1 à l'indice c si c+1 est un candidat
valide pour la case en question et 0 sinon ;
- le nombre de candidats restants - pour ne pas avoir à le recalculer à chaque fois ;
- le troisième indique le seul chiffre possible lorsqu'il n'y a qu'un candidat valide
 et 0 dans tous les autres cas ;
- les coordonnées, ligne et colonne, de la case dans la grille dont elle provient */

typedef struct CASE{
  char candidats[DIM];
  char nb_candidats;
  char chiffre;
  char lig; // ligne d'une case dans une grille
  char col; // colonne d'une case dans une grille
} CASE;
```

```
/* une grille est un tableau carré dont chaque élément est une case */
typedef CASE GRILLE[DIM][DIM];
```

Programmation modulaire.

Le programme comprendra plusieurs **fichiers sources** (*.c), accompagnés de **fichiers d'en-têtes** (*.h). Je propose pour commencer :

- un fichier `constantes.h` qui regroupera les constantes utilisées dans le projet
- un fichier `structures.h` qui regroupera les structures définies
- un fichier `outils.c`, accompagné de son fichier d'en-tête `outils.h` qui regroupera les fonctions de traitement des grilles et des cases
- un fichier `sudoku.c` qui comprendra le programme principal.

Ces fichiers devront être convenablement **commentés** :

- chaque fichier devra comprendre un en-tête indiquant son auteur, sa date de création, etc
- chaque définition de type ou de fonction sera précédée d'un commentaire en bloc indiquant son mode d'utilisation, son rôle, ses limites, etc
- le code proprement dit sera accompagné de commentaires en ligne ou en fin de ligne décrivant le rôle et la signification des variables, des instructions, des conditions, des blocs, etc

Les fichiers d'en-têtes seront protégés des inclusions multiples.

Vous écrirez un `Makefile` pour faciliter la compilation et l'édition de liens entre vos différents fichiers.

Exercice 1 :

1. Dans un dossier `Sudoku/programmes`, créez les fichiers `constantes.h`, `structures.h`, `outils.c`, `outils.h` et `sudoku.c` et répartissez-y les différentes fonctions et en-têtes de fonctions en n'oubliant pas les directives d'inclusion.
2. Tracez le graphe de dépendances des différents modules et créez le fichier `Makefile`.

Exercice 2 : Quelques fonctions sur les cases :

1. `void affiche_case(CASE c)` affiche proprement chaque champ d'une case. Utile pour la mise au point des autres fonctions.
2. `void init_case(CASE *c, char ligne, char colonne, char chiffre)` initialise une case ; la variable chiffre peut prendre une valeur comprise entre 0 et `DIM` ; les variables `ligne` et `colonne` prennent des valeurs comprises entre 1 et `DIM`.

Exercice 3 : Quelques fonctions sur les grilles :

1. `void saisie_grille(GRILLE g, char *adr)` saisit une grille à partir d'un fichier :
   - chaque ligne du fichier correspond à une ligne de la grille ;
   - les "chiffres" peuvent être ¿ 9 mais ils doivent être séparés par un (ou plusieurs) espace ;
   - le second argument doit être une chaîne de caractère indiquant l'adresse du fichier ;
   - pas de contrôle sur la correction de la saisie.
2. `void affiche_grille(GRILLE g)` affiche une grille sous forme d'un tableau de chiffres sans séparateurs :
   - une case remplie est représentée par le chiffres correspondant ;
   - une case non remplie est représentée par 0 ;
   - cette fonction n'est pas adaptée au cas où les nombres inscrits dans les cases ont plusieurs chiffres.
3. `void affiche_grille2(GRILLE g)` produit un affichage un petit peu plus sophistiqué dans lequel chaque sous-région est délimitée (mêmes limitations que la fonction précédente).
4. `void enreg_grille(GRILLE g, char *adr)` sauve une grille sous forme de fichier texte, sous un format qui permet de la recharger.